AD-A120 095 TEXAS UNIV AT AUSTIN DEPT OF COMPUTER SCIENCES F/G 5/2
ANNUAL SCIENTIFIC REPORT FOR GRANT AFOSR-81-0205, 15 JUNE 1981 --ETC(U)
JUL 82 K M CHANDY. AFOSR-81-0205
UNCLASSIFIED AFOSR-TR-82-0880 NL

1 OF 1
AD A
120095

END
DATE
FILMED
11-82
DTIC

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A120095

3

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0880	2. GOVT ACCESSION NO. AD-A120 095	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ANNUAL SCIENTIFIC REPORT FOR GRANT AFOSR-81-0205, 15 JUNE 1981-14 JUNE 1982		5. TYPE OF REPORT & PERIOD COVERED ANNUAL, 15 Jun 81-14 Jun 82
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) K.M. Chandy		8. CONTRACT OR GRANT NUMBER(s) AFOSR-81-0205
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department University of Texas Austin TX 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE July 1982
		13. NUMBER OF PAGES 11
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

DTIC ELECTE OCT 1 2 1982

A

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In the last year, work on the grant was carried out in three areas: (1) the development of distributed algorithms to detect termination of distributed computations, (2) methods for proving the correctness of distributed programs, with an emphasis on proving temporal properties and (3) the development of distributed algorithms to solve graph problems. Work in all three areas was fruitful, resulting in new ideas and refereed papers in technical journals and conferences. The author has felt that the key issues in designing and proving distributed software are (1) proving that all the processes in the system cooperate in main- (CONT.)

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: taining a system-wide invariant and (2) developing algorithms by which asynchronous processes can determine collectively that the system as a whole has entered certain states. Work in the last year was based on the above premise. The author's work on developing new distributed algorithms supports this premise.

AFOSR-TR- 82-0880

Annual Report for Grant AFOSR 81-0205

Research Objectives

The goal of this project is to study the problem of developing correct and efficient distributed software, i.e. software which consists of cooperating processes. The specific focus is on the distributed nature of the distributed software. Our objective is to extend techniques developed for sequential programs to distributed programs.

The development of a sequential program consists of posing assertions, constructing a program to ensure that these assertions are maintained and then proving that they are. Termination is shown by demonstrating that execution of the program reduces some metric which is bounded from below. The major problems in extending this methodology to distributed systems are: (1) no one process can assert, unilaterally, that an invariant holds, because some other process may cause the invariant to be violated and (2) a distributed system may terminate in the form of a deadlock for instance, even though no process in the system has terminated; furthermore, no process can assert unilaterally that the entire system has terminated. The fundamental problem with distributed systems is to ensure cooperation among processes in maintaining invariants and in achieving proper termination. Therefore, the focus of this project was, and continues to be, issues of cooperation.

82 10 12 166

The project has been extremely successful in the last year resulting in the identification of fundamental problems and the publication of solutions to some of these problems. Work was carried out in 3 areas:

1. Distributed algorithms for detecting termination of distributed computations.

2. Methods for proving correctness of distributed software.

3. The development of distributed algorithms to solve problems in various application areas.

Termination Detection

A distributed computation may terminate due to a deadlock or because the computation has been successfully completed. The major impetus for developing distributed termination detection algorithms has come from distributed data bases where the concern is to detect deadlock (because data bases may be presumed to run indefinitely). Therefore our primary goal last year, was to develop correct, practical and simple distributed algorithms to detect deadlock. Motivation for attacking the problem was also derived from the following statement in a recent paper by Gligor and Shattuck: "Renewed interest in distributed systems has resulted in the publication of at least ten protocols for deadlock detection. However, few of these protocols are correct and fewer appear to be practical."

In a system consisting of processes which only communicate with a single central agent, deadlock can be detected easily because the central agent has complete information about every process. Deadlock detection is more difficult when there is no such central agent and processes may communicate directly with one another. If we could assume that message communication is instantaneous or place other restrictions on message delays, deadlock detection becomes simpler. However, the only realistic general assumption is that message delays are arbitrary (but finite). We present deadlock detection algorithms for networks of processes in which there is no single central agent and in which message delays are arbitrary (but finite). We only assume that messages sent by a process A to a process B are received by B in the order sent by A.

We consider two models of deadlock in message communicating systems: resource and communication deadlocks. Deadlock detection algorithms are given for both models. Most models of deadlock in distributed data bases are resource deadlock models [5,6,7,8,9,10,11,14]; in these models deadlock arises because processes may wait permanently for one another for resources held by each other. The communication deadlock model is a more abstract and more general model of deadlock; it is applicable to any message communicating system of processes.

We have presented and proved the correctness of simple, practical algorithms to detect resource and communication deadlocks [1,2,12].

Methods for Proving Correctness of Distributed Software

Our primary goal in this area has been to extend well-known sequential programming proof constructs such as pre-condition, post-condition and the use of metrics in proving termination to distributed programs. The obvious advantage in using techniques which are extensions of sequential-programming techniques is that the tools and the experience gained from sequential programming can be applied to distributed programs as well. We work with a general model of distributed systems; we do not require that distributed programs be coded in any particular language for purposes of proof.

The key features of our method are:

1. Modular Specification: We present a scheme for specifying processes in a modular fashion. The specification relies exclusively on a process's interaction with its environment and is independent of process implementation.

2. Hierarchy: We present inference rules by which a specification for a network is derived from specifications of component processes. Thus the proof of a network is not concerned with implementations of component processes.

3. Compatibility With Sequential Programming Proof Techniques: We have extended well known sequential programming proof constructs such as pre-condition,post-condition and the ideas of termination proof to distributed systems. Those familiar with the Floyd-Hoare proof technique for sequential programming should find our method to be straightforward.

We use some ideas from sequential program proofs in proofs of message-passing systems. In an annotated proof of a sequential program, each statement s has a precondition pre(s) and a postcondition post(s). The proof shows that if assertion pre(s) holds prior to execution of s, post(s) holds following execution of s assuming execution of s terminates. We shall use the precondition/postcondition concept for describing process safety properties. Proofs of liveness (or termination) in sequential programs are based on demonstrating the existence of a metric such that the execution of each statement causes the metric to decrease in value. We will use a similar technique in process proofs. However, processes can wait indefinitely for messages, something that conventional sequential programs do not do; to handle this we introduce a new concept called activity which is the condition under which a process will definitely send or receive a message. Other liveness properties are derived from the basic property of activity and from safety.

We have developed a coherent extension of sequential programming proof techniques to distributed programs. Several examples are found in Ossefort [15].

The Development Of Distributed Algorithms To Solve Problems In Various Application Areas

We have attempted to develop distributed algorithms in two application areas: simulation and graph problems. The application areas were chosen because of their importance and the familiarity of the principal investigators with these areas. Our pioneering effort in the distributed simulation area has received wide recognition; therefore our effort last year was primarily in graphs. By developing distributed simulations to important problems we hoped to gain experience in writing and proving distributed programs, as well as making a contribution to the literature on algorithms.

We began by developing a distributed solution for one of the most-studied problems in graphs: finding the shortest path between vertices. A distributed solution is important in the following situation: communication paths are being set up between processes in an unreliable, and perhaps even hostile environment. Since no process has information about all other processes in the network, centralized, sequential-programming algorithms cannot be used.

We developed a distributed algorithm to detect shortest paths in graphs which have negative cycles. We also demonstrated the application of our shortest-path algorithm in solving a variety of graph problems including depth-first search.

Another important problem in graphs is that of detecting knots: a vertex in a directed graph is in a knot if for every vertex v_j reachable from v_i, v_i is also reachable from v_j. The problem of knot detection is important because of its relevance to deadlocks [3,4]. We developed a scheme whereby a vertex (which is represented by a process) can determine if it belongs to a knot [13].

Computing Network-Wide Functions

We found that there was a sizable class of problems with the following structure:

Processes in a network cooperate in computing a result which we call the global-result where

$$\text{global-result} = f(\text{local-result}(i), \text{ for all processes } i)$$

where local-result(i) is some computed result in process i, at its termination, and f is any computable function. The knot detection problem is only one of many problems that fall within this class. We developed general solutions to solve this class of problems. Our solution was proved correct and its application to specific practical problems was demonstrated.

SUMMARY

The past year has been very productive. If we can continue the same rate of productivity in the future we shall be very pleased. In the future we plan to enter new areas and to ensure that the results of the past year are accepted and used by the computer sciences community.

REFERENCES

[1] Chandy, K.M., J. Misra and L. Haas, "A Distributed Deadlock Detection Algorithm and Its Correctness Proof," to appear Communications of the ACM.

[2] Chandy, K.M. and J. Misra, "A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems," Proceedings of the ACM SIGACT-SIGOPS Principles of Distributed Computing Conference, August 18-20, 1982, Ottawa, Canada.

[3] Chang, Ernest, "Decentralized Deadlock Detection in Distributed Systems," University of Victoria, Victoria, British Columbia, Canada V8W 2Y2.

[4] Dijkstra, E.W. and C.S. Scholten, "Termination Detection for Diffusing Computation," Information Processing Letters, Vol. 11, No. 1, pp. 1-4, August 1980, North-Holland Publishing Company.

[5] Gligor, V.D. and S.H. Shattuck, "Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, SE-6, 5, September 1980, pp. 435-440

[6] Goldman, B., "Deadlock Detection in Computer Networks," Technical Report MIT LCS-TR185, M.I.T., September 1977.

[7] Gray, J., "Notes on Database Operating Systems," in Lecture Notes in Computer Science, Springer-Verlag, 1978.

[8] Isloor, S.S. and T.A. Marsland, "An Effective 'On-Line' Deadlock Detection Technique for Distributed Database Management Systems," Proceedings COMPSAC 1978, IEEE, pp. 283-288.

[9] Lomet, D.B., "Coping with Deadlock in Distributed Systems," Research Report RC 7460 (#32196), IBM T.J. Watson Research Center, December 1978.

[10] Mahoud, S.A. and J.S. Riordon, "Software Controlled Access to Distributed Databases," INFOR 15, 1, February 1977, pp. 22-36.

[11] Menasce, D. and R. Muntz, "Locking and Deadlock Detection in Distributed Databases," IEEE Transactions on Software Engineering, SE-5, 3, May 1979, pp. 195-202.

[12] Misra, Jayadev and K. M. Chandy, "Termination Detection of Diffusing Computations in Communicating Sequential Processes," Transactions on Programming Languages and Systems, Vol. 4, No. 1, January 1982, pp. 37-43.

[13] Misra, J. and K.M. Chandy, "A Distributed Graph Algorithm: Knot Detection," to appear Transactions on Programming Languages and Systems.

[14] Obermarck, R., "Distributed Deadlock Detection Algorithms," ACM TODS, Vol. 7, No. 2, June 1982.

[15] Ossefort, Marty, "Correctness Proofs of Communicating Processes - Three Illustrative Examples from the Litera[illegible]e," to appear Transactions on Programming Languages and Systems.

Annual Report (AFOSR 81-0205)

List of Publications

1. Distributed Computation on Graphs: Shortest Path Algorithms, to appear in Communications of the ACM, (K. M. Chandy and J. Misra)

2. A Distributed Deadlock Detection Algorithm and Its Correctness Proof, to appear in Communications of the ACM, (K. M. Chandy, J. Misra and L. Haas)

3. A Distributed Graph Algorithm: Knot Detection, to appear in ACM Transactions on Programming Languages and Systems, (J. Misra and K. M. Chandy)

4. A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems, Proceedings of the ACM SIGACT-SIGOPS Conference on the Principles of Distributed Computing, August 18-20, 1982, Ottawa, Canada. (K. M. Chandy and J. Misra)
5. Proving Safety and Liveness of Communicating Processes with Examples, Proceedings of the ACM SIGACT-SIGOPS Conference on the Principles of Distributed Computing, August 18-20, 1982, Ottawa, Canada (J. Misra, K. M. Chandy and Todd Smith)

Annual Report (AFOSR 81-0205)

List of Professional Personnel

K. Mani Chandy, Principal Investigator

Jayadev Misra, Faculty, Computer Sciences, UT Austin